Q&A for Assignment 2
(Kui Wu)

Q: Do we need to consider non-TCP packets?

Ans: No, non-TCP packets, if any, should be filtered out, by checking the Protocol field in the IP header.

Q: For Start and End time, are you wanting the timestamp of the connection (for example the first packet has a timestamp of Feb 6 2006) or are you wanting the time in regards of the cap file (thus, the start time of the first packet would be 0.000000)?

Ans: Use relative time, i.e., the time with respect to the cap file. Please print out the unit of time as well (in either ms or in second).

Q: How to count the start time and end time of a complete TCP connection? For part B of output, does time start at state S1Fx and end at state SxF2 or SxF1?
Also for a "complete" connection does that refer to a connection that successfully completes a three-way handshake (state: S2F0) or only connections that connect and close (state: S2F2)?
Ans: The start time is the time when the first SYN is seen for this connection. The end time is the time when the last FIN is seen for this connection. To make it simple, you should ignore whether or not this SYN or FIN has been successfully ack-ed. For a complete connection, the time starts at state S1F0, and ends at SxF2 (but if SxF2 cannot be found in the trace, it ends at SxF1).

"Complete TCP connections" are DEFINED IN THE ASSIGNMENT as those that we see at least one SYN and at least one FIN in the trace file.

Q: When we count the number of RST connections, do we count the amount of times a RST flag was set or how many connections had one or more RSTs. For example, lets say a connection had 3 RST flag sets, would that be considered 1 reset connection or 3 reset connection? I guess in simplest terms, are we counting the RST flag or connections who had at least one RST occurred? (I hope this question makes sense!)

Ans: We count the number of connections that had one or more RSTs. In your example, if a connection had 3 RST flag sets, it is considered as one reset connection.

Q: Is a packet with RST bit set considered a packet sent from source to destination?

Ans: RST could happen on both sides (i.e., in packets from source to dest as well as in packets from dest to source).

Q: Just to be absolutely certain, it's safe to assume that (for the purposes of this assignment) the lower-level headers that must be removed will always be Ethernet II and IPv4 headers?

Ans: Yes.

Q. Can we assume that a given 4-tuple will only be used for a single connection? i.e. if a connection ends, that 4-tuple used will never be used again to start a new connection that needs to be reported separately in our list of connections?

Ans: Let's assume that the 4-tuple UNIQUELY identify a connection. The chance that the same t-tuple is used by one TCP connection first (open-close) and then by another TCP connection is close to zero.

Q: Possible connection states include all combinations of S#F# where # can be 0-2, as well as R, and nothing else, correct? The document says "and so on" when listing them, but it is not entirely clear what other states may be possible. And is this "S#F#"/"R" value what is supposed to be reported for "status" in the output?

Ans: "and so on" means S#F# where # could be other numbers such as 3, 4, or even 10 (if the connection stay for a long time).

Q: I'm having trouble fully understanding the usage of the RST flag in a TCP connection. All the sources I've read give very vague answers "Reset the connection" and similar unhelpful explanations.

A 'reset' implies that the connection will be reinstated, however it doesn't seem like the RST flag carries that implication.

For example, in our sample-capture-file, the first connection starts with SYN, SYN/ACK, ACK. A proper 3-way handshake to establish connection. Then there are several ACKs back and forth with data flowing from the server to the client. Everything as expected. Then the server (port 80) sends an ACK/FIN packet signalling they are done sending data. The client (port 1200) sends back an ACK (with the expected ack num), but doesn't send its own FIN. It then sends a second packet with the ACK/RST flags set. This is the final packet in the connection.

Why didn't the client respond to the ACK/FIN with their own ACK/FIN? What's the purpose of the RST packet in this context? And finally, would this be considered both a complete connection and a reset connection? You mentioned that to be considered complete, at least 1 SYN and at least 1 FIN must be sent in the connection. Both of these are met. But the existence of the RST flag bothers me.

Ans: Whenever a host TCP gets confused on the TCP state (e.g., old duplicate SYN), it can send a RST segment so that the sender and the receiver can resynchronize again. In other words, RST does not necessarily imply disconnect and reinstated.

RST has been used in many context, including aborting a connection, since some TCP implementation uses RST instead of normal FIN to abort a connection, which sometimes is called "abortive release."

The following post "The many ways of handling TCP RST packet" is interesting to read:

https://www.snellman.net/blog/archive/2016-02-01-tcp-rst/

The good news is that Assignment 2 does not need you to mess up with the "many ways of handling RST packet".

Note again that "Complete TCP connections" are DEFINED IN THE ASSIGNMENT as those that we see at least one SYN and at least one FIN in the trace file. A complete connection can also include RST segment, and in this case it is a complete connection as well as a reset connection.

Q: How to calculate the values regarding RTT in output Part D?

Ans: Assume that you have three complete TCP connections, C1, C2, C3. Use RTT as an example. Assume that you obtained RTT values for C1, C2, C3 as follows:
C1: {t11, t12, ... t1n}

C2:{t21, t22, ... t2m}

C3: {t31, t32, ... t3l}.

Then for Minimum RTT value, you should output min{t11, t12, ... t1n, t21, t22, ... t2m, t31, t32, ...t3l}

For Max RTT value, you should output max{t11, t12, ... t1n, t21, t22, ... t2m, t31, t32, ...t3l}

For mean RTT value, you should output mean{t11, t12, ... t1n, t21, t22, ... t2m, t31, t32, ...t3l}.

Be very careful about the mean calculation. The following way in calculating the mean is WRONG:

[(t11+t12+...+t1n)/n + (t21+t22+...+t2m)/m + (t31+t32+...+t3l)/l] /3

Q : How to calculate RTT? It is so complicated, because when some segments get lost we do not know the match between data and ack.

Ans: Keep in mind that calculating RTT is not rocket science, and thus there is no such thing called "absolutely" correct. All we care is to have a reasonably accurate estimation. The two methods introduced below are good enough for practical use. Either way is acceptable for the assignment.

Q: What is exactly the algorithm for finding RTT in a TCP connection?

Ans: Use the following figure as an example. Note that the trace file is captured at the client side. As such, for RTT estimation, we only need to consider three cases: (1) the data from the client and the corresponding ack from the server, (2) the SYN from the client and the corresponding ACK from the server, (3) the FIN from the client and the corresponding ACK from the server.

You have two ways to estimate RTT (both ways are acceptable in the assignment). **I use case (1) as the example.**

Method 1:

Scan the trace file top bottom. For each data packet, find the FIRST matching ACK (i.e., ACK # = Seq # + data size in bytes). Use the matched packet pair to calculate RTT. Any packet that cannot find a match will be ignored.

In the example, the packet (seq=100) matches the packet (ACK=200). Use the pair of the packets to calculate RTT. The packet (seq=200) and the packet (seq=300) will not find any match. The packet (seq=400) will find a match (ACK=500) and the pair will be used to calculate RTT. The second Seq=200 will not find any match neither.

Discussion: Method 1 is simple, but it has the pitfall that it may slightly overestimate RTT. In the example, the second packet Seq=200 should match the packet ACK=500 for RTT calculation.

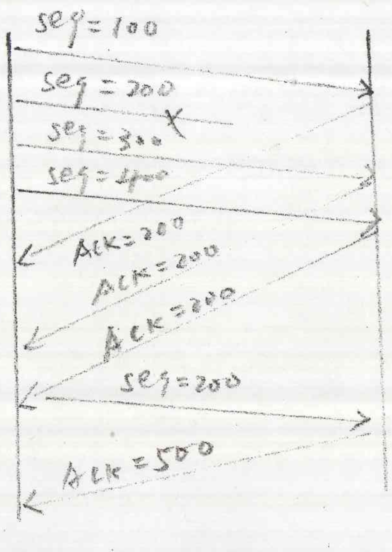
Due to its simplicity, however, it is acceptable in Assignment 2 if you implement in the above way.

Method 2:

Scan the trace file bottom up. For each ACK packet, ignore this ACK packet if there are more ACK packets that have the same ACK # in the same connection. Find the FIRST data packet that has the smaller Seq # in the same connection. Use the matched packet pair to calculate RTT. Any packet that cannot find a match will be ignored.

In the example, packet ACK=500 will find a match (i.e., scan from bottom up, the first Seq= 200 data packet). Use the matched packet pair to calculate RTT. **From Bottom up**, the first Packet ACK=200 and the second packet ACK=200 will be ignored, because there is a third packet having ACK=200. Use the last packet ACK=200 (bottom up) to find a matching packet (which is Seq=100). Use the matched packet pair to calculate RTT.

Discussion: Method 2 is slightly difficult to implement, but it may have more accurate estimation. Nevertheless, there is no guarantee that it always returns "correct" RTT value. For instance, it is wrong if the connect is reset during the packet Seq=100 and the packet ACK=500. Nevertheless, in the assignment you do not need to consider TCP reset in the RTT calculation.



Assume that the size of data = 100 bytes